JSC 38608

Deorbit Flight Software Demonstration Project Summary

ENGINEERING DIRECTORATE

AEROSCIENCE AND FLIGHT MECHANICS DIVISION

15 January 1998

National Aeronautics and
Space Administration

Lyndon B. Johnson Space Center
Houston, TX

Deorbit Flight Software Demonstration Project Summary

Prepared By:

____

Denise M. DiFilippo
G. B. Tech, Incorporated

Approved By:

______________________________ ______________________________

David A. Petri
GN&C Rapid Development Lab Manager
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

James P. Ledet
Code Q RTOP Project Manager
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Concurred By:

______________________________ ______________________________

Aldo J. Bordano, Chief
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Karen D. Frank, Chief
GN&C Development & Test Branch
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

This Page Intentionally Blank

Table of Contents

Figures

Tables

Acronyms and Abbreviations

AFMD	Aeroscience and Flight Mechanics Division
API	Application Programming Interface
COTS	commercial, off-the-shelf
EGI	Embedded GPS/INS
FCOS	Flight Control Operating System
FSSR	Shuttle Flight Subsystem Software Requirements
FY	Fiscal Year
GN&C	Guidance, Navigation & Control
GPS/INS	Global Positioning System/Inertial Navigation System
HIL	hardware-in-the-loop
ISO	International Standards Organization
IV&V	Independent Verification and Validation
JSC	Johnson Space Center
NASA	National Aeronautics and Space Administration
OMS	Orbital Maneuvering System
RDL	Rapid Development Laboratory
SLOC	Source Lines of Code

This Page Intentionally Blank

1.0 Introduction

The National Aeronautics and Space Administration (NASA) is studying the feasibility of making major upgrades to the Space Shuttle, including the Shuttle avionics. A decision to upgrade the avionics hardware could carry with it the need to upgrade the onboard flight software and the associated ground based test, training and support software and hardware systems.

What is the best overall approach if such a software system upgrade is required? One possibility would be to take advantage of this opportunity to rewrite the flight software using modern tools, technology and techniques. It is difficult to accurately evaluate the potential impact of such a large and unique effort, in costs, schedules and risks, although it certainly would be a very large and complex endeavor.

The Aeroscience and Flight Mechanics Division (AFMD) at NASA's Johnson Space Center (JSC) Engineering Directorate is exploring ways of producing Guidance, Navigation & Control (GN&C) systems more efficiently and cost effectively. A significant portion of this effort is software development, integration, testing and verification.

A natural synergy thus exists, in which AFMD could apply their evolving facilities and techniques while providing empirical data to the analysis of effort and risk involved in upgrading the Orbiter flight software.

AFMD has established the GN&C Rapid Development Laboratory (RDL), a hardware/software facility designed to take a GN&C design project from initial inception through hardware-in-the-loop (HIL) testing and perform final GN&C system verification. The operations approach for the RDL emphasizes the use of commercial, off-the-shelf (COTS) software products to implement the GN&C algorithms in the form of graphical data flow diagrams, to automatically generate source code from these diagrams and to execute the software in a real-time, HIL environment, following a Rapid Development methodology.

The Flight Software Demonstration Project constructed a core team of systems, software, and GN&C domain experts. The team identified a significant yet manageable subset of the Space Shuttle GN&C onboard flight software, the deorbit flight phase, to implement and demonstrate. Making use of the facilities and philosophy of the RDL, the GN&C application software was designed, implemented, and tested. Multiple software engineering environments were explored and compared. Areas where the RDL processes and infrastructure needed augmentation or improvement were identified, and these evolved along with the flight software itself, throughout the project. The resulting flight software was released for Independent Verification and Validation (IV&V), and has been widely demonstrated to NASA management. The final demonstration included pilot-in-the-loop capability and OMS actuator in the loop, with the flight software executing on a target flight computer.

2.0 Project Description

2.1 Scope

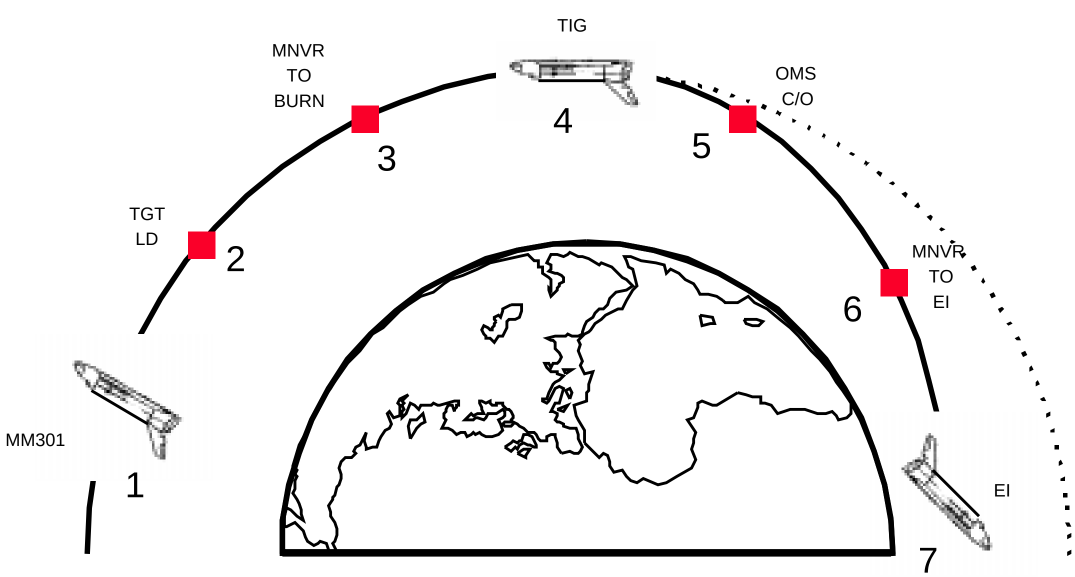
The team limited the scope of this effort to implementing the GN&C functions for a single mission phase only. The deorbit phase was chosen. This phase had not previously been addressed in the RDL, so it would increase our knowledge and add to our library of core functions, while providing a blank development slate for a relatively pure test of the capabilities of the approach.

While the software is planned to have wider application to general vehicle deorbit, Shuttle operations were chosen as the target environment, both to provide input to the Orbiter Upgrades project and to take advantage of a great deal of specific data pertaining to shuttle operations already available to the team. The deorbit phase, in Shuttle operations, is from the selection of the Deorbit/Entry software (Ops 3) to Entry Interface.

The operations demonstrated correspond to three major modes of Shuttle operations. These are MM301 (Deorbit Coast), MM302 (Deorbit Execution), and MM303 (Pre-Entry Monitor).

Figure 1. Deorbit GN&C Event Timeline

Event	Description	Elapsed Time (typical)
1	MM 301 - Simulation Start	0:00
2	Deorbit Target Load	0:30
3	Maneuver to Burn Attitude	1:30
4	OMS Burn Ignition	5:00
5	OMS Burn Cut-Off	9:03



The design and implementation assumes little modification of the current Shuttle GN&C Functional Subsystem Software Requirements (FSSRs). The primary change is the assumption of the availability of GPS/INS (Global Positioning System/Inertial Navigation System) sensor data, which reduces the state propagation requirements. Also, no alignment from the star tracker was included.

Subsets of all various flight functions are implemented in order to demonstrate and assess the process for all components of a GN&C software build. A detailed presentation of the functions and capabilities provided can be found in JSC-38601, *GN&C Deorbit Flight Software Description Document (Demonstration Project)*.

The objectives of the project were:

- Prototype an improved process for flight software development and verification
- Develop an initial set of “Next-Generation” flight software for Shuttle Orbiter Upgrades
- Demonstrate software commonality and evaluate multiple tools, languages, target platforms and real-time operating systems
- Execute GN&C Flight Software on candidate Orbiter Upgrades avionics architecture
- Demonstrate code quality through Quality Assurance and Independent Verification and Validation assessments

2.2 Deliverables

The following are the major deliverables developed for this project:

- Shuttle Orbiter GN&C application flight software source code and executable
- environment simulation
- application program interface
- configurable real-time test environment
- advanced displays
- alternate flight software (using BEACON)

Other significant project accomplishments included:

- developing a single-string, avionics flight computer prototype using commercial parts and standards
- integrating and executing the hardware, simulation, and flight software with a pilot-in-the-loop, in real-time
- inspecting and documenting the software
- performing unit, module and system level testing
- performing independent validation and verification of the software
- putting fundamental software infrastructure in place for multiple projects

2.2.1 Flight Software

The flight software was mostly developed using the ISI MatrixX tool suite (including Xmath, SystemBuild and Autocode). Source code was generated from MatrixX block diagrams, using the autocoder. The autocoder can produce both C and Ada code. For this project, code was primarily generated in C, and the C code version was the one that went through formal testing and IV&V. (Some versions were also coded in Ada and tested informally; though time limitations precluded exhaustive testing, preliminary results indicated that the C and Ada autocode are comparable.) Some functions were written directly in C, and then imported into the MatrixX environment as “User code blocks”.

The flight software developed for this project includes GN&C functions for Space Shuttle major modes (MM) 301 (Deorbit Coast), 302 (Deorbit Execution), and 303 (Pre-Entry Monitor), which encompass operations from Ops 3 to Entry Interface. The software supports both automatic and manual (pilot-in-the-loop) capabilities.

For requirements, the team assumed that we would reproduce the functionality specified in the related FSSRs, with three exceptions. Navigation functions would be written to take advantage of the availability of GPS/INS sensors. Some interfaces to other subsystems would not be implemented, since this is a stand-alone GN&C deorbit module demonstration. And The application software was run under VxWorks real-time operating system, using an auto-generated real-time scheduler instead of the Orbiter Flight Control Operating System (FCOS).

The system was developed in evolutionary cycles, with each cycle demonstrating end-to-end functionality at increasing levels of fidelity, quality and confidence. The major cycles and their descriptions are summarized in table 1 below.

.

Table 1. Project Cycles

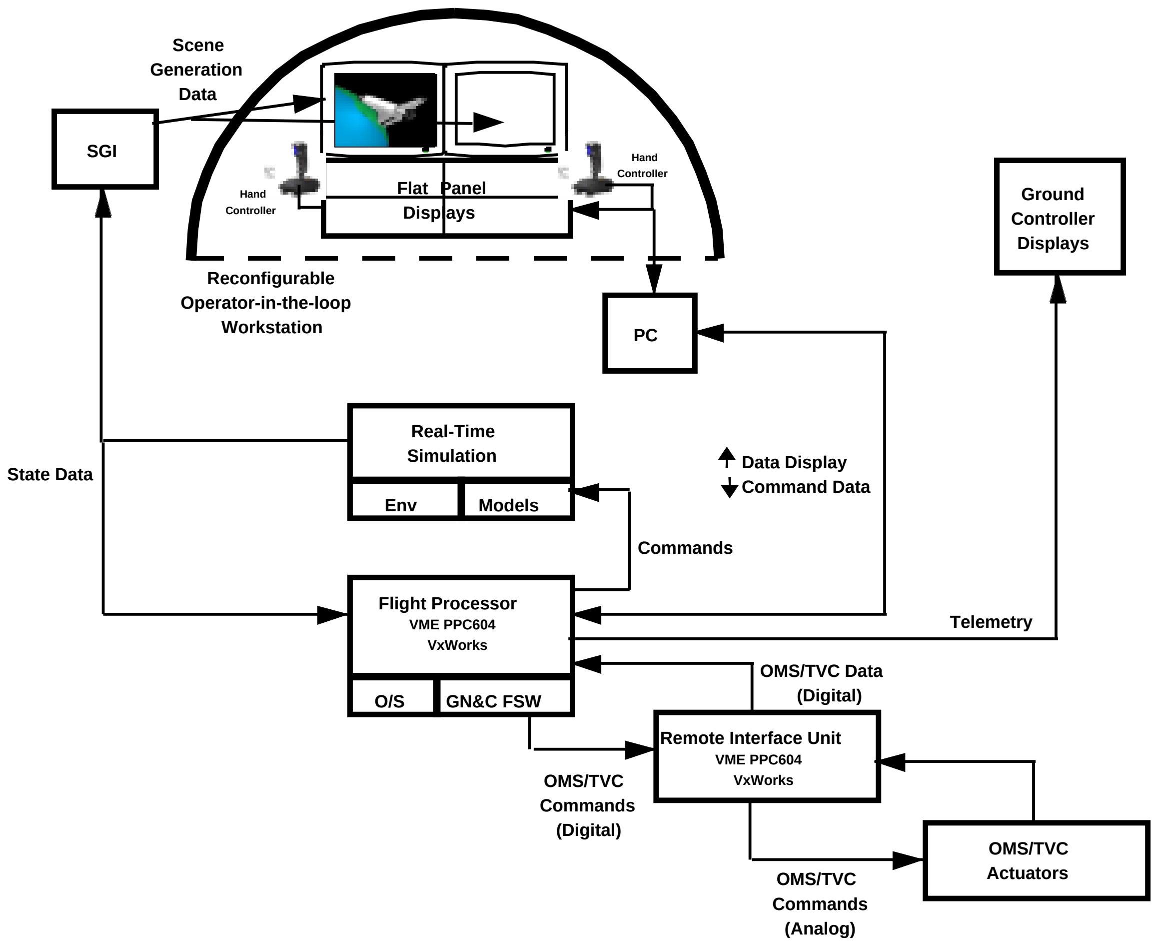
Date Completed	Version	Description
11/06/96	0.1	Preliminary release; minimum functionality; emphasized system architecture and connectivity, system integration of closed loop flight software, and design simulation
12/08/96	0.2	Preliminary release of partially functional flight software to support more extensive platform and autocoder tests and integration of the flight software with the high fidelity real-time simulation
12/20/96	0.5	Intermediate release, including full automatic mode flight software functionality; automatic nominal capabilities
03/19/97	0.6	Full functionality (within scope) development completed, including manual modes, limited OMS RM, and support of full deorbit maneuver display. This version underwent formal inspections. Also used to support development of test and verification scripts.
03/31/97	0.65	Same functionality as version 0.6. Inspections completed. All major defects (identified in inspections) corrected. This version used for component testing.
06/06/97	0.7	All remaining defects (identified in inspections) corrected. Component testing completed and defects corrected. This version used for integrated subsystem and system testing.

Table 1. Project Cycles

Date Completed	Version	Description
09/12/97	0.8	Subsystem and system testing completed and defects corrected. This version released for IV&V
12/17/97	0.9	IV&V completed and defects corrected.

The flight software load is a complete, stand-alone system capable of flying the GN&C deorbit modes identified, both in automatic and manual flight configurations. The real-time software interacts with the environment simulation, and has been demonstrated, using a reconfigurable test workstation, with both hardware and pilot in the loop. In the mock-up, the flight software sends data to the displays, and takes input from touch screens, keyboard, and a hand Controller.

Figure 2. Deorbit Flight Software Test Configuration



2.2.2 Environment Simulation

Testing with a high fidelity real time environment simulation is critical to the ability to verify and validate the flight software and document its quality.

The environment models include the vehicle dynamics, and the sensor and effector hardware, as well as physical models of the planet (e.g. gravity and atmosphere).

Two distinct simulations were developed for this project; a low fidelity development simulation and a high-fidelity simulation for use in formal testing and demonstrations.

The low fidelity development simulation was built in the MatrixX System Build environment. It was used to support closed-loop simulation within the MatrixX environment. This simulation supported developer testing of the software prior to autocoding and integration testing.

The high fidelity simulation used the TRICK simulation environment, which was developed and is supported by the Automation, Robotics and Simulation Division (ER) at JSC. TRICK is an environment shell that takes as input environment models written in C and builds an executable simulation environment. Associated with TRICK is a library of models that previous and current users developed and then made available for reuse. (At the beginning of this project, the models available in the TRICK library were primarily on-orbit models, since this is where TRICK had seen the greatest application to date.)

TRICK was installed in the RDL, and members of the core team became familiar with it. New models were developed as needed to support the deorbit environment. In particular, OMS (Orbital Maneuvering System) and EGI (Embedded GPS/INS) models were developed, and other effector and sensor models were augmented as needed. The ENIGMA graphics library was ported to the RDL as well. The team also integrated a new prototype display, created by the JSC RAPIDS lab with human factors engineering from Space and Life Sciences.

Both batch and real-time simulation capabilities are available. The real-time simulation has been integrated with a reconfigurable operator-in-the-loop workstation and with advanced real-time displays.

The environment models that were developed will be used to augment existing model libraries and be available for reuse in other RDL projects and by other TRICK users.

2.2.3 Application Programming Interface

An important feature of our rapid development paradigm is that, as hardware becomes available, it is included in the test loop. Thus, for example, the deorbit flight software initially ran with simulated operator inputs, but as our test environment evolved we were able to substitute actual real-time operator inputs. OMS actuators were also initially simulated but later actual hardware was included in the test loop. As another example, early versions ran on general purpose workstations but later software was ported to a real-time platform. Eventually the real-time workstation and hardware and operator in the loop environments were integrated via reflective memory.

To facilitate multiple, evolving operational and test environments, an Application Programming Interface (API) was developed. By standardizing the interface, moving among configurations is relatively transparent to the flight software. As new equipment is added to the environment, most of the changes are limited to adding new modules to the API.

The API is used to partition the application code from the communication interfaces. This partitioning, coupled with the use of POSIX coding standards, allows the flight software to be easily ported and interfaced to the operating system, databuses, and hardware. This well-defined interface allowed easy switching among different databuses (e.g., reflective memory, ethernet), workstations, hardware, etc.

2.2.4 Reconfigurable Mock-up

The RDL reconfigurable mock-up includes compartments and connections for a variety of equipment useful in a piloted test and simulation environment. These include workstation displays, data displays, touch screens, keyboards and hand-controllers, which all can be linked into the RDL real-time network to set up the desired test, simulation or demonstration.

Much of the current capability and flexibility of this facility evolved with this project.

2.2.5 BEACON version of Flight Software

BEACON is an alternative flight software automatic code generation tool under consideration for use in the RDL. To get some idea of the relative merits of Beacon, a parallel implementation effort was undertaken using Beacon.

The Beacon development effort was less comprehensive than the MatrixX effort in several respects. No formal documentation effort was performed. No formal quality assurance functions (e.g., inspections) were performed. Testing was informal and not comprehensive, compared to the MatrixX version. IV&V was not performed.

The Beacon effort was able to leverage off of the MatrixX effort as well, since a significant portion of the MatrixX effort had been completed before beginning development with Beacon. Significant understanding and clarification of requirements, based on the existing FSSRs, had already been achieved by the team and was not repeated for Beacon implementation. Overall architecture, as well as test cases, test data, and detailed logic (for example, jet select logic) were also carried over to this development effort.

Data handling in the two languages proved to be significantly different. MatrixX is more data flow oriented; data are passed primarily via calling arguments to software modules. The Beacon environment uses global data structures in order to make data available to the modules.

As of this writing, the BEACON version of the Flight Software has been completed, but not fully tested in the real-time environment.

3.0 RDL Process and Infrastructure Advances

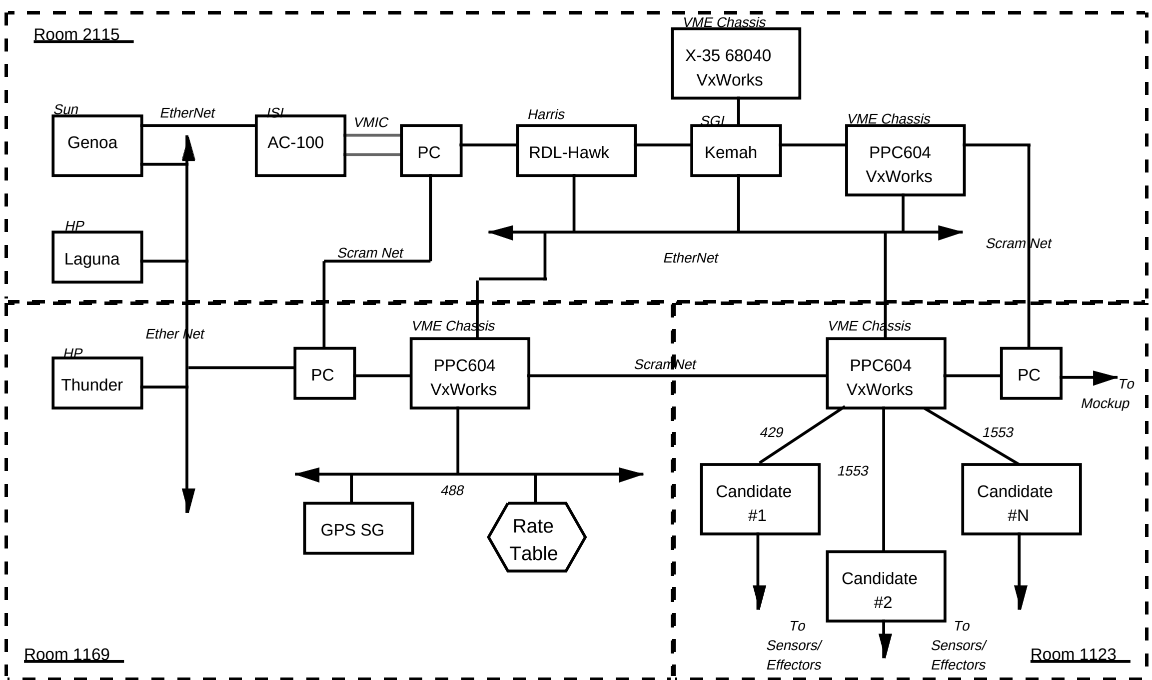
The long term vision for the Rapid Development Laboratory is to be a center-of-excellence for rapid hardware and software development, and to be the facility of choice for real-time, hardware-in-the-loop simulation and end-to-end system verification, with special emphasis on guidance, navigation and control.

This project helped us to better understand the infrastructure issues remaining in order to meet this vision. Over the course of, and in conjunction with, this project, several enhancements to RDL infrastructure were achieved, as described below.

3.1 Using the RDL Real-Time Network

A recurring theme in the evolving RDL infrastructure is the improved availability and use of tools and documents on-line over the lab network, accessible directly from each engineering workstation. Internal RDL and project specific applications and web pages are integrated with tools and processes to improve the ease, access and accuracy of project information. The implementation of a reflective memory ring and the use of commercial products and standards (e.g., databuses, operating systems) contribute to the usefulness and adaptability of the RDL.

Figure 3. GN&C RDL Network Architecture



3.1.1 Web Based Test Documentation

Test requirements were written and put on the project web pages. The test cases are also available on the web, and are cross referenced, through hyperlinks, to the test requirements.

As tests are run, the output files from the tests are transferred to an area which is also cross referenced to the test plan. An oversight script is available which can sample these sites on demand and produce a current report of test status, including completion status, test results, and pending actions. The test requirements, test plan, test data and status reports are available to all team members.

3.1.2 Configuration Management

The project team chose the COTS product ClearCase to use for configuration management. ClearCase runs on the RDL workstations, and is accessible across the lab. It readily accommodates the evolutionary nature of Rapid Development, tracking all versions. The test result files are also cross referenced to the managed version numbers.

Because the tool is so well integrated with the development environment, there is very little lag between when a piece of code is completed and when it is under configuration management and available to the rest of the team.

The flight software is currently under configuration management. For the modules developed using MatrixX, this includes both the MatrixX blocks and the code generated by the MatrixX autocoder. Plans are to expand the process to include the environment simulation, documentation, and test data under configuration management as well.

3.1.3 Error Reporting and Tracking

The COTS tool ClearDDTS was used for error reporting and tracking. It too runs on the workstations and is accessible across the lab. Testers can enter discrepancy data from the same terminal where they do the tests. When discrepancies are reported, electronic mail automatically notifies those who need to know about the error. DDTS integrates well with ClearCase, so that errors and corrections are correlated with software versions. Standard and customized error reports are available and easy to generate. Current status information is available to the team on demand. Data can be entered, tracked and reported across multiple projects.

3.1.4 On-Line Metrics Data Collection

Project team members report their project hours, by work category, directly to the metrics data base using an application developed in the RDL. A web page gives access to the data collection form. Once a team member has entered data, it is automatically included in the data base from which reports are generated. (See sections 5 and 6 for additional information regarding the metrics program and data collected.)

3.1.5 Documentation

Current versions of all project documents are available through the project web pages.

3.2 Inspections

Formal inspections were performed on the flight software.

This was a new process both in the RDL and to most members of the project team. To start the effort, the team was assisted by outside personnel who were experienced with performing inspections. A process and forms were developed. Team members received a brief training overview.

The initial inspections were organized and moderated by the experienced trainers. This responsibility was quickly taken over by the development team.

Much of the code was generated by an autocoder, from MatrixX blocks. Inspections were performed on these blocks, not the generated source code. More traditional code inspections were performed for those modules that were not developed in MatrixX, primarily the API code.

Inspection reports are available on the project web pages.

3.3 Testing Tools

We identified and investigated several advanced tools for improving test capability.

X-ATAC is a test coverage evaluation tool developed by Bellcore. Team members were trained in its use. A pre-release copy was provided by Bellcore and installed in the RDL. Unit tests for most modules included this useful analysis. The analysis shows the percent of code that was executed during a test suite. It also can show specifically which lines of code were not executed and can be used to determine what additional tests can best improve coverage.

The McCabe toolset analyses code complexity in a variety of ways (e.g., cyclomatic complexity, essential complexity, module design complexity, and others). To effectively use these tools requires considerable understanding of both the theory and implementation of complexity analysis. A demonstration copy of the toolset was obtained and installed in the RDL. An initial analysis was performed on a subset of the flight software. It was concluded that additional investigation of the toolset is recommended in order to establish guidelines and procedures for using these tools. For example, which complexity measures are important for flight software? What complexity values are desirable? What actions are recommended if measures exceed desirable levels?

Another toolset, Logiscope, was identified and investigated. The RDL does not own this tool or have a demonstration version installed. However, the SR&QA group at JSC has a copy, is familiar with its use, and assisted project team members in running some analysis of our software using Logiscope. The tool functionality of Logiscope appears approximately the same as for the McCabe toolset.

What is clear is that, in a Rapid Development environment, these and possibly other advanced tools can be powerful development aids as well as test analysis tools. A long term goal is to identify the most effective types of tools, establish guidelines and procedures for their use, and give them to developers for use in improving quality of software as it is developed. We want to develop quality as much as possible and, in the long run, depend less on testing the quality in after development.

4.0 IV&V (Independent Verification and Validation)

To help evaluate the quality of the flight software, Independent Verification and Validation (IV&V) was performed. The verifiers were requirements analysts and performance verifiers from the Shuttle Flight Software performance and verification organization. The effort focused on closed loop testing of the deorbit software. The time available for IV&V was severely limited; three testers began with limited familiarity with the software and simulation, and completed their analysis in one month.

The results of this effort should be considered more preliminary than definitive. Still, the testers were able to make significant observations and recommendations in addition to discovering several discrepancies in the flight software.

Among the general observations and recommendations are:

- Incorporating additional testing capability into the simulation and flight software, including enhanced ability to insert specific test scenarios, would be very useful
- several areas of the flight software that were considered out-of-scope for this project, may in fact be more difficult implementation and testing challenges. These include, for example, timing requirements, complete crew interfaces, sequencing and multipass operations, and redundancy management.
- areas that should be tested further include off-nominal crew operations, data values at singularities, and timing variances
- testers desired a better interface for changing parameters and collecting data

The IV&V effort identified a total of 21 discrepancies. Among these, 11 were associated with the flight software. (The remainder were related to the simulation or user interface.) Among the flight software discrepancies, there were 0 at level one (most severe), 5 at level 2, and 6 at level 3.

Overall, the testers concluded that “the application of the autocode generator to the Shuttle deorbit flight phase has had a good start.” The IV&V effort was documented in “Space Shuttle Programs, Level 7 GN&C Flight Software Performance Verification, Independent Verification and Validation of Deorbit Software” (Hickey, Ware, Zophy & Mills; Lockheed Martin; September 30, 1997), which is available on the project web pages

5.0 Metrics Program

To begin a metrics program in the RDL, we concentrated on collecting development engineering metrics. (A long term goal is to add sustaining engineering and project management metrics.) Details of the objectives, strategy, selection and definitions for the metrics program plan can be found in JSC-38605, *Guidelines for the Rapid Development of Software Systems*.

For this project, the following table shows the metrics that were collected.

Table 2. Metrics Collected

Metric Classification	Description
Software Size	The SLOC in the system that must be tested and maintained
Software Size	MatrixX Block Counts
Software Size	Size of executables
Software Staffing	Number of engineering and first line management personnel involved in system development
Development Progress	Number of modules successfully completed from design through test
Software Performance	Execution times
Test Case Completion	Percent of test cases successfully completed
Discrepancy Report Open Duration	Time lag from problem report initiation to problem report closure
Fault Density	Open and total defect density over time
Design Complexity	Number of modules with a complexity greater than an established threshold

For some of these measurements, we do not yet have sufficient historical data base to know how or if they will prove useful for planning or managing a Rapid Development project, or for evaluating its quality. An example is MatrixX block counts. By collecting this data, we hope to learn more about effective metrics in a Rapid Development environment.

6.0 Metrics Results

The metrics data collected for this project is presented in the following figures.

Figure 4. Flight Software Source Lines of C Code

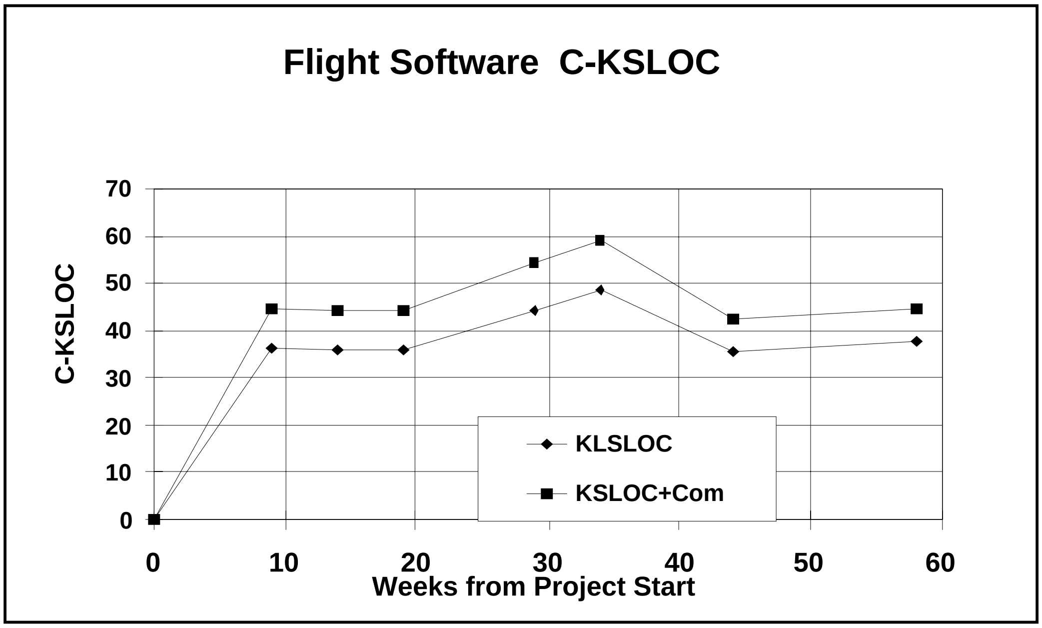


Figure 5. Flight Software plus Simulation SLOC

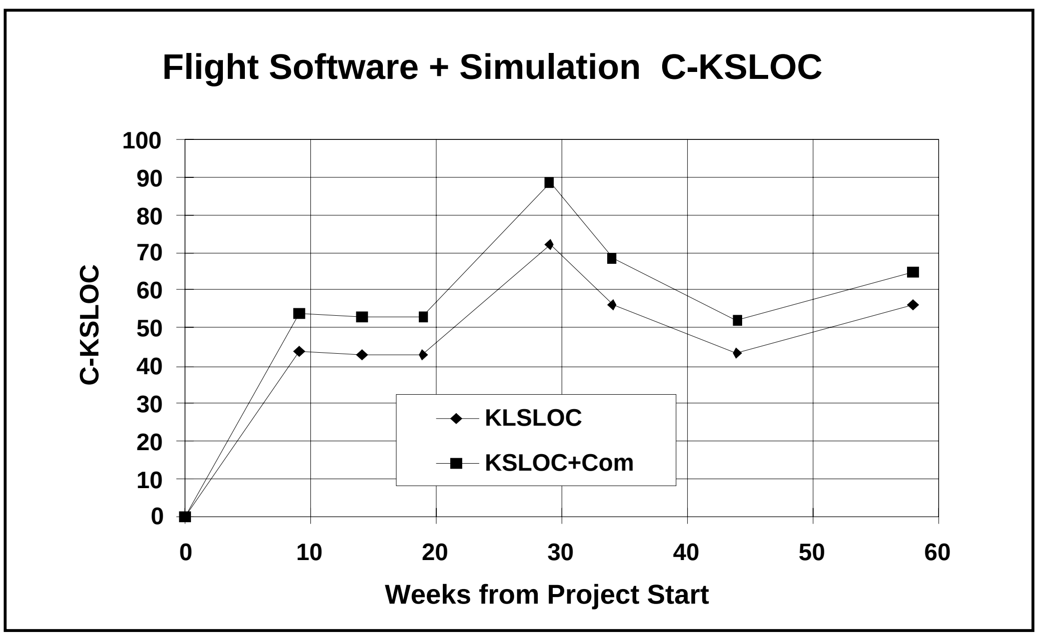


Figure 6. MatrixX Block Counts (Flight Software)

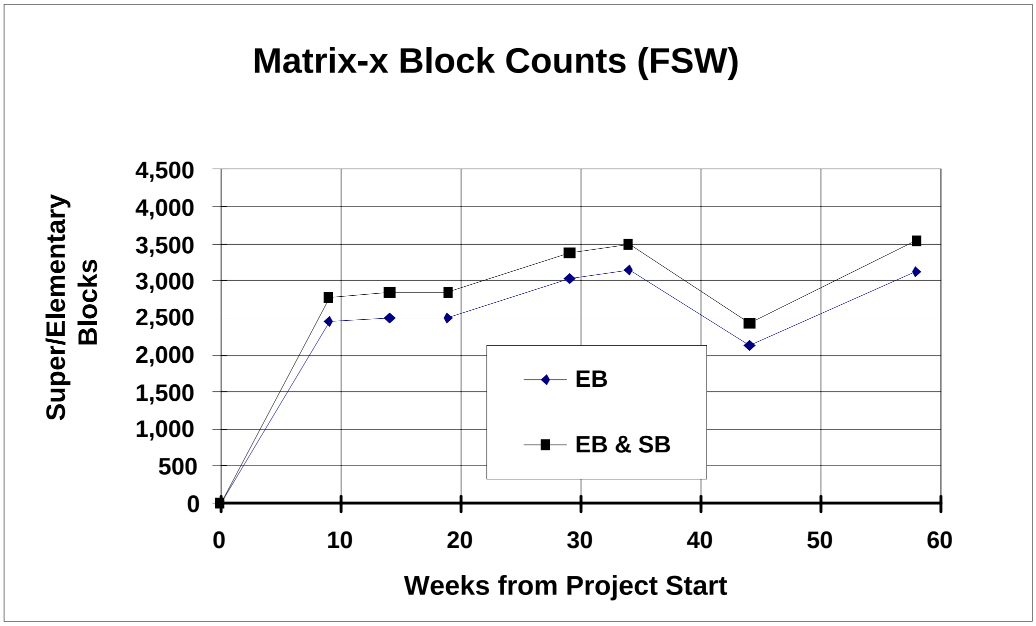


Figure 7. MatrixX Block Counts (Flight Software plus Simulation)

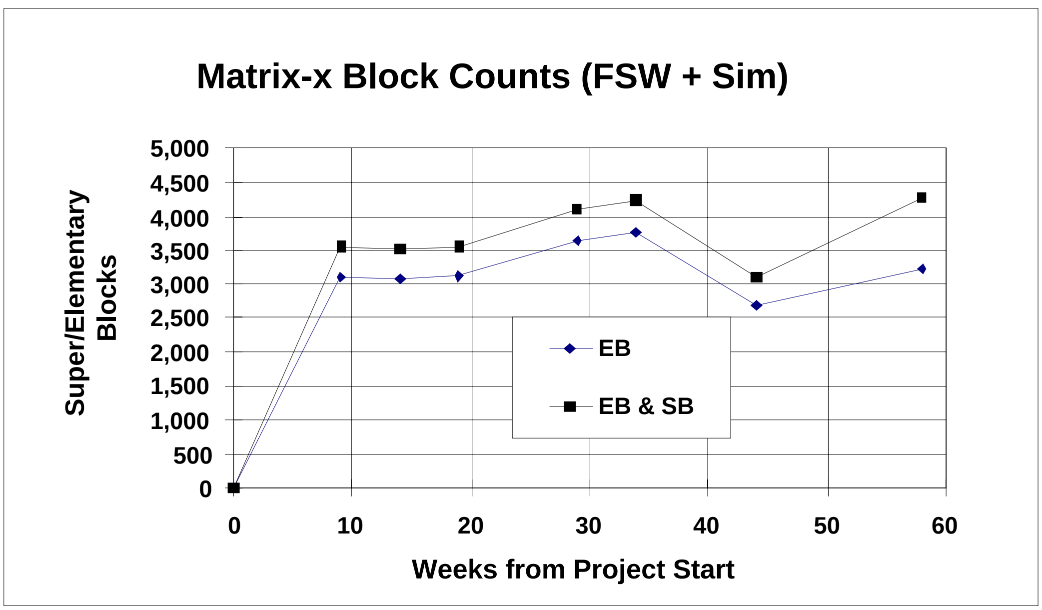


Figure 8. Project Staffing Hours

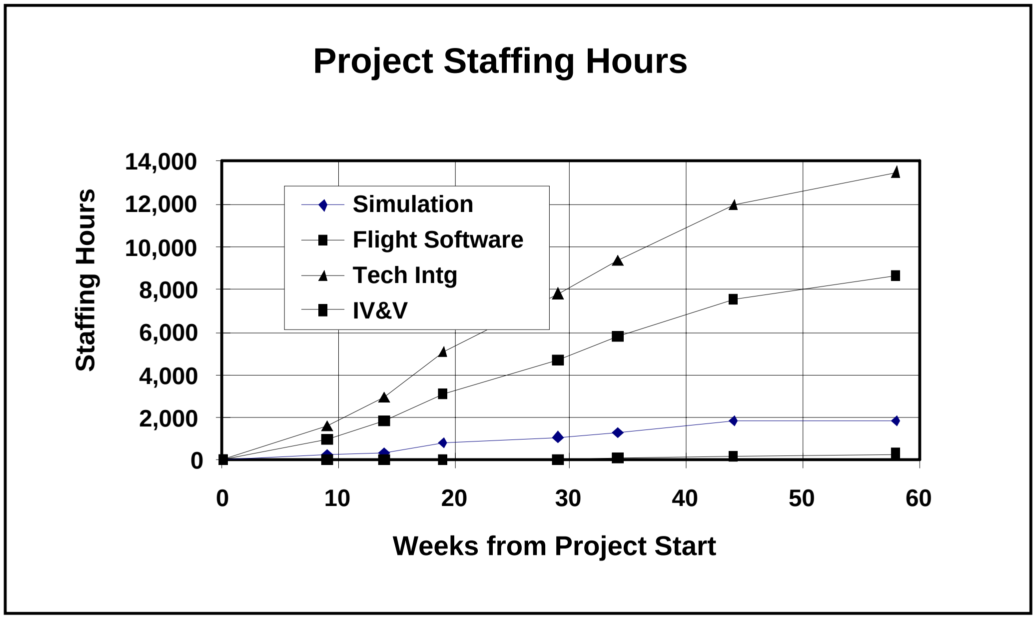


Figure 9. Development Progress

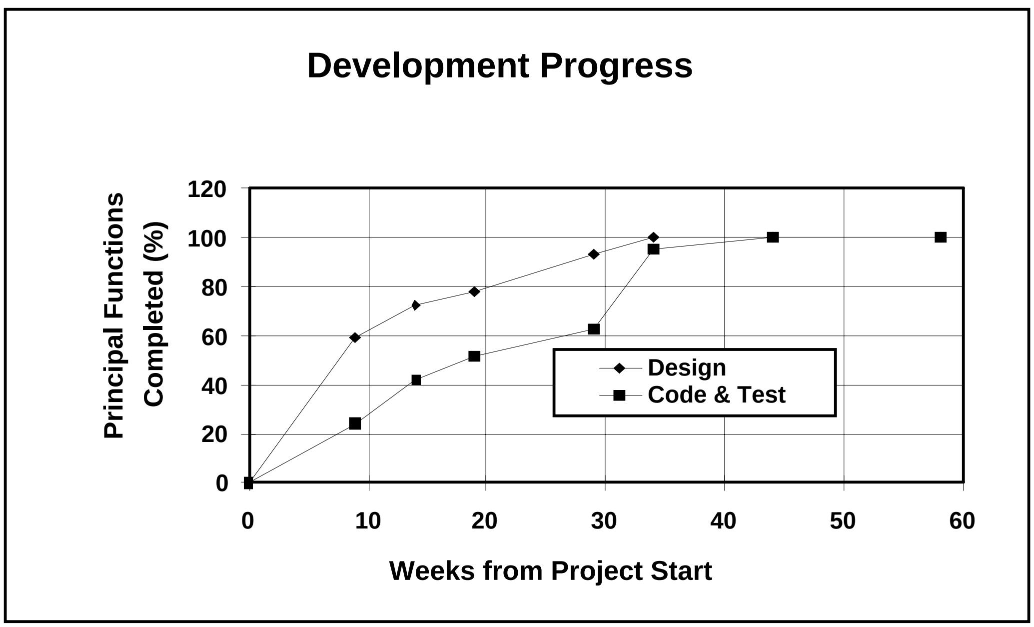


Figure 10. Guidance Module Run-Time Performance

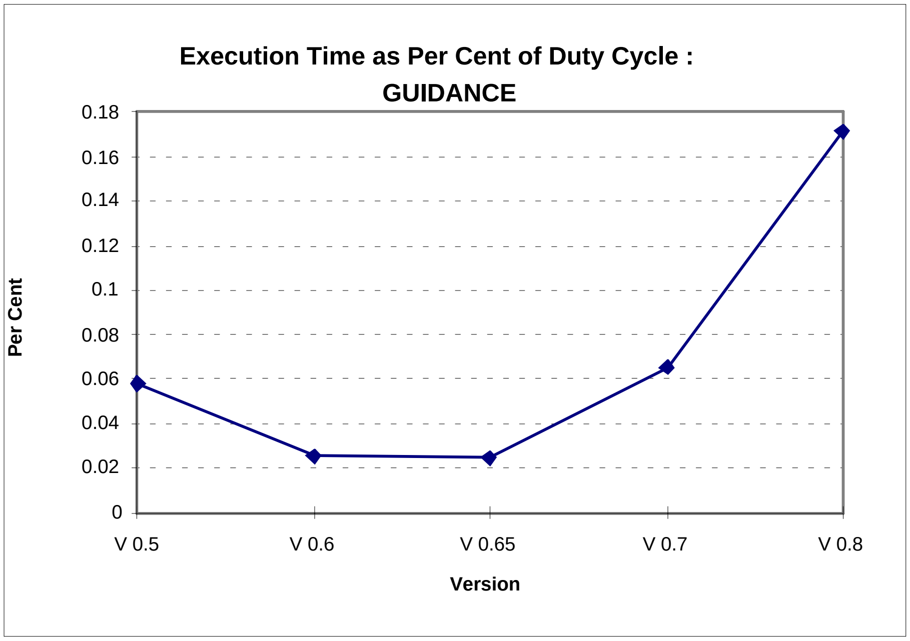


Figure 11. Control & Navigation Run-Time Performance

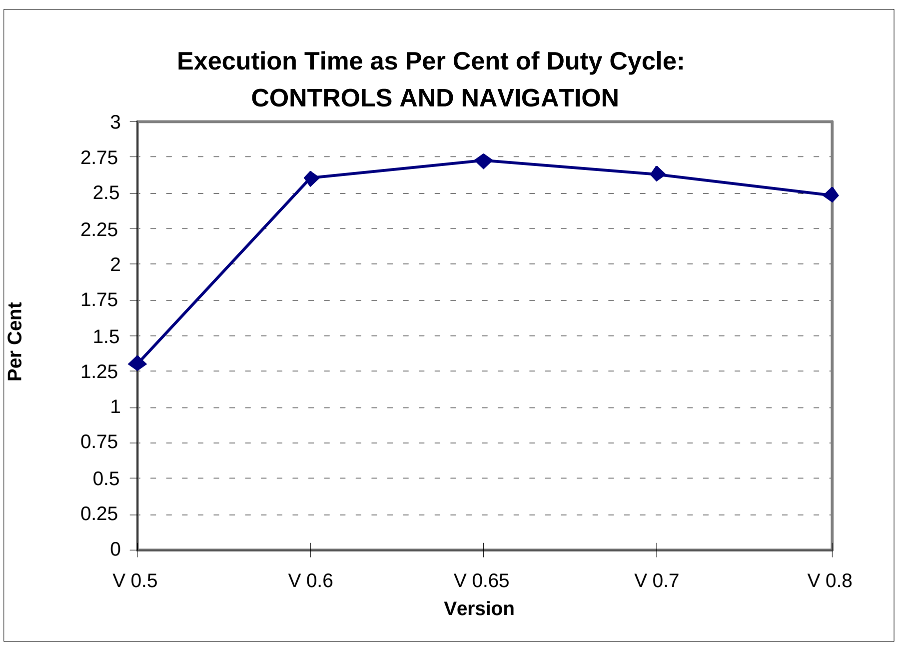


Figure 12. Size of Executable Flight Software (1 of 2)

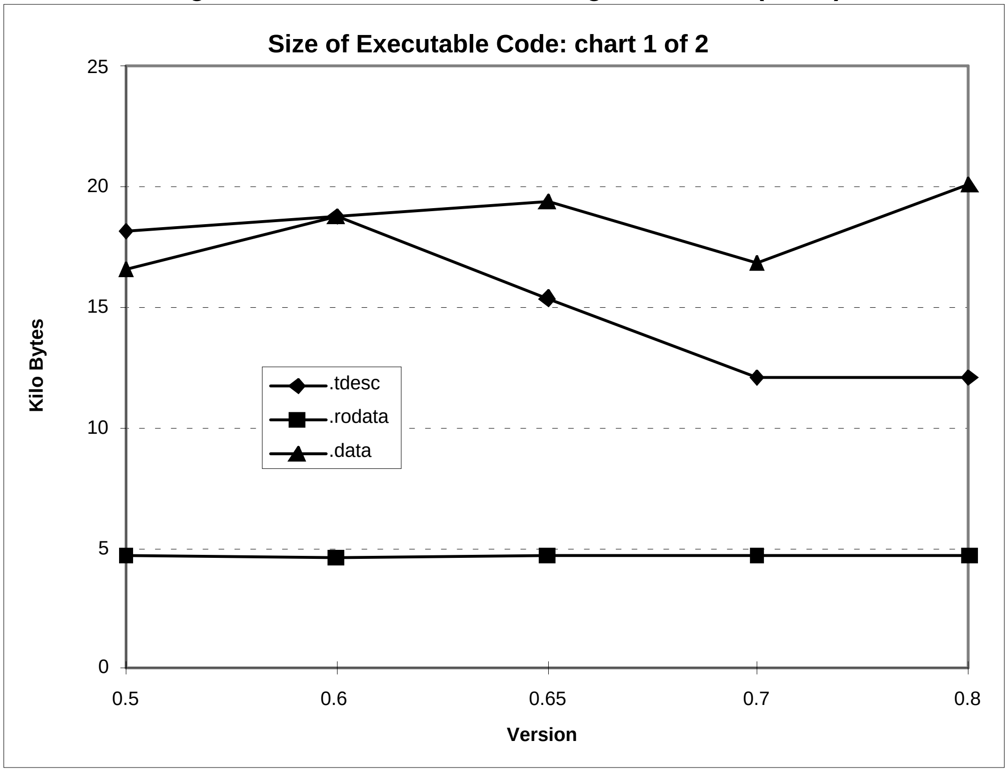


Figure 13. Size of Executable Flight Software (2 of 2)

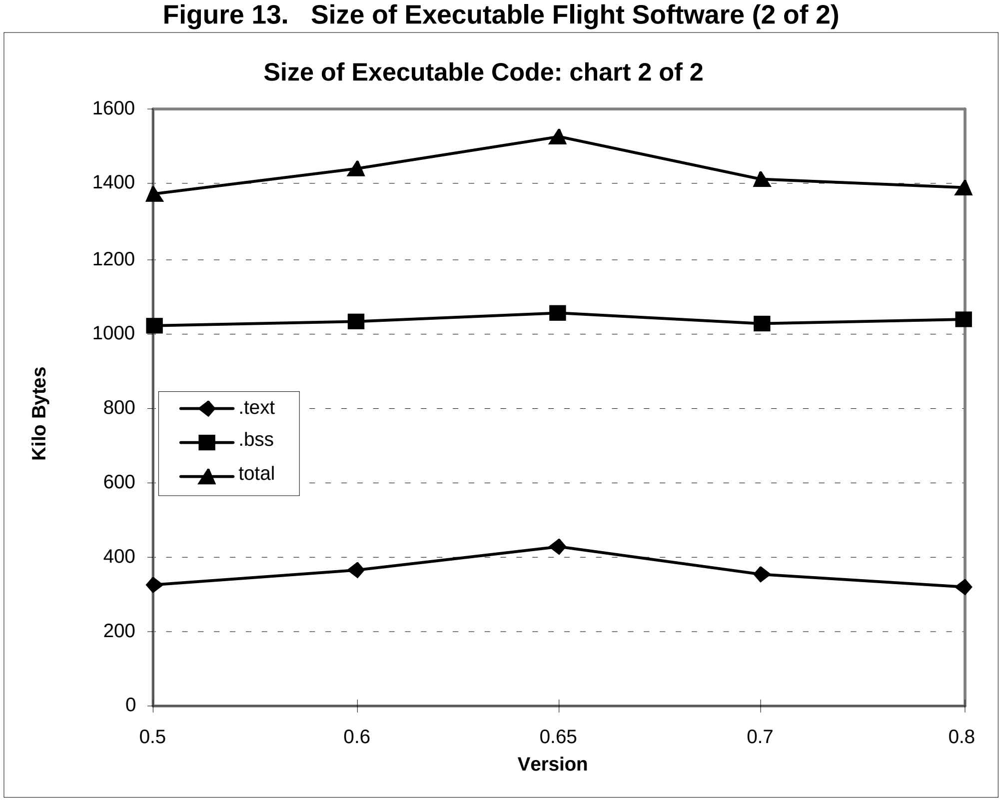


Figure 14. Discrepancy Report Open Duration

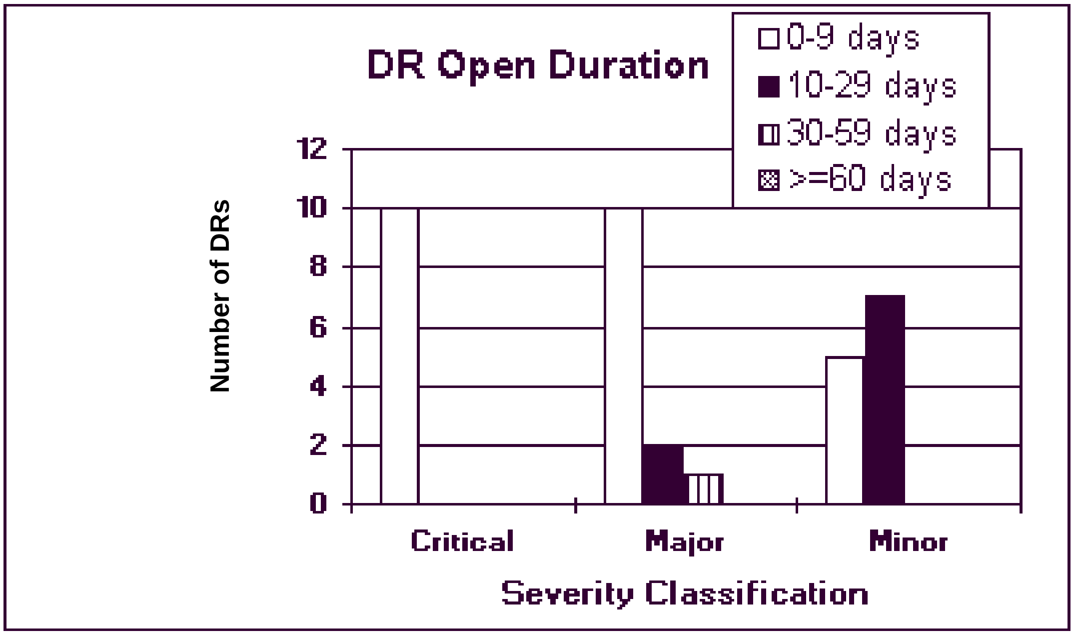


Figure 15. Discrepancy Report Status

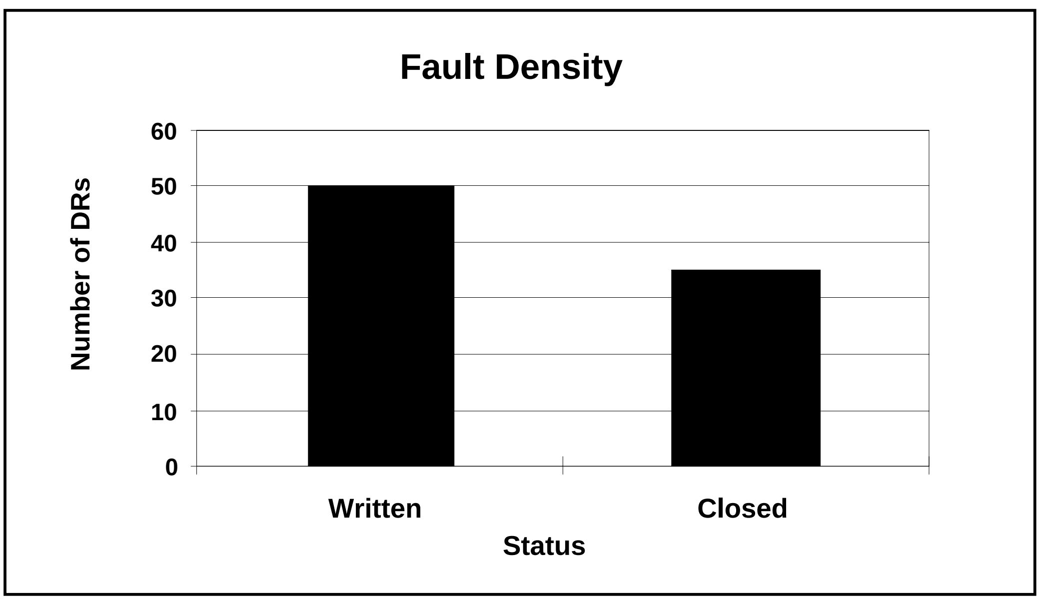
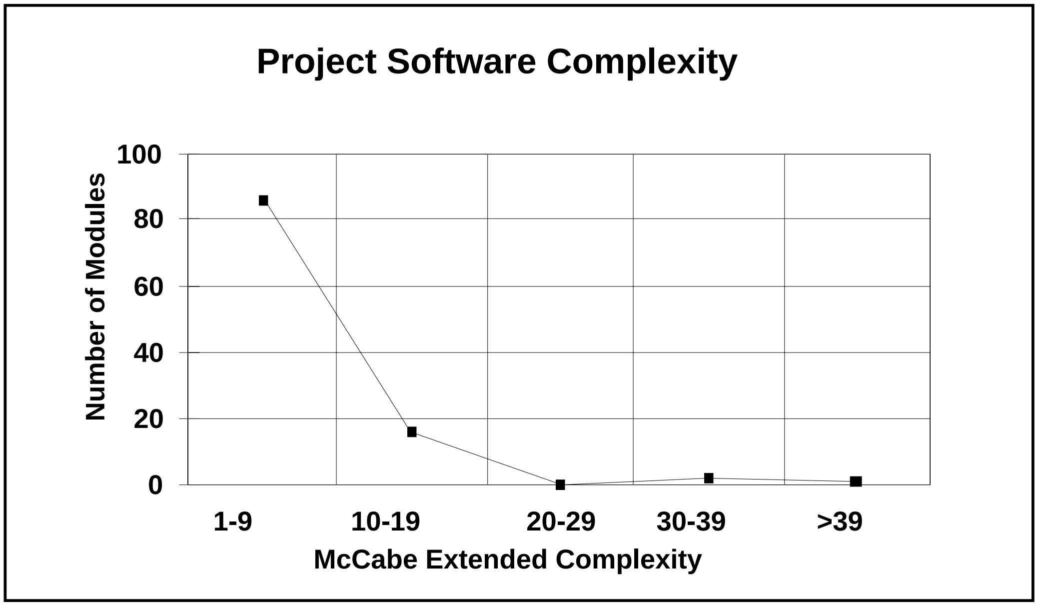


Figure 16. Software Complexity



7.0 Conclusions

Recall that the stated project objectives were to:

- Prototype an improved process for flight software development and verification
- Develop an initial set of "Next-Generation" flight software for Shuttle Orbiter Upgrades
- Demonstrate software commonality and evaluate multiple tools, languages, target platforms and real-time operating systems
- Execute GN&C Flight Software on candidate Orbiter Upgrades avionics architecture
- Demonstrate code quality through Quality Assurance and Independent Verification and Validation assessments

This is an ambitious set. Yet all objectives were met and demonstrated by the project deliverables. Clearly this project did not give definitive results in all of these areas. Just as clearly, significant progress was made. The first of the stated objectives is the most difficult to address with certainty.

Prototype an improved process for flight software development and verification

We began with a set of guidelines for rapid development, and expanded and improved on them as the project required and our experience suggested. There is no doubt that the team exercised a new methodology for this project, and that our efforts have improved on it. An improved process should be better, faster, and/or cheaper in some quantifiable way. Our engineering judgement supports the hypothesis that this method is an improvement over traditional software development methodology when used to develop flight software. To a certain extent, our metrics empirically support this conclusion.

One measure of the cost of the project is staff hours expended. Using this measure, the total cost of developing the Deorbit GN&C flight software was 13,993 man hours (through Cycle 0.8) distributed over flight software, infrastructure and test rigs (simulation support), technical management (Training, Programmatics, Configuration Management, Process, Technical Meetings, et al.). This estimate does not include time allocated to the IV&V activities which are expected to add an additional 900 to 1,400 man hours to the project cost, resulting in an estimated total of about 15,000 to 15,500 man hours to develop approximately 56,000 lines of code. To compare this to a conventional software development paradigm the software metrics tool CHECKPOINT was used to estimate the cost to develop 56,000 lines of code subject to the constraints shown in the table below. The CHECKPOINT metrics tool is commercially available software marketed by Software Productivity Research to provide cost estimation and other metrics support capabilities derived from a database of over 6,000 scientific, industrial, and commercial projects.

Using these parameters, CHECKPOINT estimates 37,212 man hours would be required to complete the project using traditional development techniques. The rapid development cost of about 15,500 man hours is a reduction of 58% compared to the conventional development cost.

Table 3. CHECKPOINT Model Input

Category	Class
Project nature	New program development
Project scope	Disposable prototype
Project Class	Internal Program, for use at a single location
Primary project type	Embedded or real time program
Secondary project type	Scientific or mathematical program
Project goals for estimating	Find the highest quality, with normal staff
New problem complexity	Many difficult and complex calculations
New code complexity	Fair structure, but with some complex paths/modules
New data complexity	Complex data elements and data interactions
New code	56,227 lines of “C” source code

Approximately 80 discrepancy reports will be written against the Deorbit GN&C flight software and API during unit, module, system and IV&V testing. This yields a defect density of about 1.4 defects per KSLOC. The CHECKPOINT data also estimates a defect density of about 13.5 defects per KSLOC. This implies an order of magnitude reduction.

These comparisons should certainly be interpreted in context; they do not prove superiority of our rapid development methodology, but they strongly suggest it. Several factors no doubt influenced these numbers in addition to development approach. For example, inefficiencies in automatic code generation may inflate the lines of code above traditional expectations, making productivity comparisons difficult. Minimal requirements and design activity was required, due to existing FSSRs, and this likely decreased actual cost compared to expectations. And additional testing might have uncovered more discrepancies. Moreover, the software is completed only as defined by the project. That is, it is not really ready to fly. However, the project also suffered from the burden of training team members in the use of the rapid development methodology, use of development and test tools, and the GN&C flight software requirements. As well, the RDL infrastructure to support the project was being determined and implemented concurrent with the project. These influences likely inflated the overall cost of the project.

Overall, the results are quite positive and support the continued evolution of the Rapid Development Laboratory as a valuable facility, and the Rapid Development Methodology as an effective approach for modern software development.